SoFiA User Manual

Version 0.5.0 (22/09/2015)

Preamble

The SoFiA user manual provides a detailed description of SoFiA's user interface and explains how to set and modify source finding parameters and invoke the pipeline.

Table of Contents

Introduction

SoFiA, the **So**urce **Fi**nding **A**pplication, is the source finding and parameterisation pipeline of the WALLABY and DINGO survey projects. Its name is derived from the Greek word **σοφία**, which means 'wisdom'. While originally designed for the detection of compact, isolated sources in three-dimensional **HI data cubes** (in short: galaxies), SoFiA is versatile enough be be used on other three-dimensional data cubes, e.g. CO data, and potentially even two-dimensional images. Another feature of SoFiA is its modern graphical user interface that allows the user to conveniently control the pipeline settings, launch the pipeline, and inspect the output catalogue.

At its core, SoFiA is a selection of Python and C++ modules that are fully controlled by a single **parameter file**. While the user interface provides a convenient way of automatically generating a parameter file based on the user's settings, parameter files are really just plain text files that can also be created manually in a text editor. Once a parameter file has been generated, the pipeline can be invoked from either the **user interface** or the **command line**. The latter option provides the flexibility to run SoFiA as part of a shell script, e.g. to automatically process a large batch of data cubes.

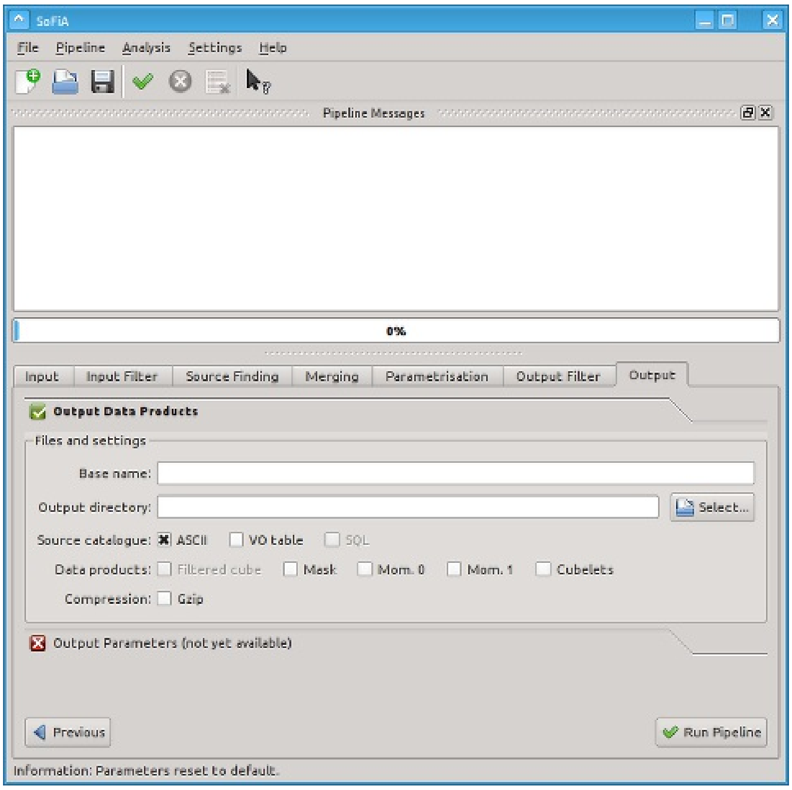
This **user manual** describes both the user interface as well as the large number of control parameters of SoFiA. It is meant to guide the user through the large set of available options and provide a basic understanding of how the different filtering, source finding and parameterisation algorithms work. Additional help on each input parameter is available through the 'What's This?' option in the user interface. It should be noted, though, that source finding is a fairly **complex** task that will strongly depend on the nature and quality of the data cube to be searched as well as the specific algorithms and settings to be used. We therefore encourage users to not rely under any circumstances on the default settings provided with SoFiA, but rather experiment with different settings and algorithms in order to obtain optimal results.

← Previous ↑ Up Next →

User Interface

Introduction

SoFiA provides a comprehensive and modern graphical user interface based on the [Qt framework](). It has been designed to run on different operating systems and provide the native look & feel on each system. Hence, a SoFiA window in Mac OS X will look slightly different from a SoFiA window on a Linux machine, but the functionality will be the same on both systems. The screenshot shown on this page was obtained on a Linux/KDE system in ‘Plastik’ style.



Elements of the User Interface

The user interface of SoFiA can be subdivided into the following areas:

- Menu Bar
- Tool Bar
- Pipeline Messages
- Parameter Settings
- Status Bar

Keyboard Shortcuts

	Mac OS X	Linux / KDE	Linux / GNOME
New File	CTRL+N	CTRL+N	CTRL+N
Open File	CTRL+O	CTRL+O	CTRL+O
Save File	CTRL+S	CTRL+S	CTRL+S
Save File As	CTRL+SHIFT+S		CTRL+SHIFT+S
Quit	CTRL+Q	CTRL+Q	CTRL+Q
Run Pipeline	F2	F2	F2
Abort Pipeline	ESC	ESC	ESC
Full Screen	F11	F11	F11
User Manual	CTRL+?	F1	F1
What’s this?	SHIFT+F1	SHIFT+F1	SHIFT+F1

← Previous ↑ Up Next →

User Interface

Menu Bar

The following tasks and options are available from the menu bar:

- **File**
  - **New** – Close the current file and reset all parameter settings to their default values.
  - **Open** – Open a file and read parameter settings.
  - **Save** – Save parameter settings to currently opened file.
  - **Save As** – Save parameter settings to a new file.
  - **Quit** – Exit from SoFiA. Any unsaved parameter settings may be lost, although SoFiA will save all current settings to a temporary session file which will be restored when SoFiA is re-launched from within the same directory. It will not be possible to exit from SoFiA while the pipeline is running.
- **Pipeline**
  - **Run Pipeline** – Start the pipeline using the current parameter settings. SoFiA will save the current parameter settings to a temporary session file before invoking the pipeline.
  - **Abort Pipeline** – Abort the current pipeline run. All pipeline processes will be terminated immediately without waiting for them to finish. Hence, the pipeline is not likely to produce any useful results in this case and will have to be restarted.
  - **Save Messages As** – Save all pipeline messages to file.
  - **Clear Messages** – Clear the pipeline message interface. This will discard all previous pipeline output messages. If you wish to keep a copy of the pipeline messages, please save them before clearing the message interface.
- **Analysis**
  - **View Catalogue** – Display the source catalogue after successfully running the pipeline. Note that the catalogue viewer will only read catalogues written in VO-compliant XML format and will not work with ASCII catalogues.
- **Settings**
  - **Show Pipeline Messages** – Show/hide pipeline message window and progress bar.
  - **Show Toolbar** – Show/hide SoFiA's tool bar.
  - **Full Screen** – Enable/disable full-screen mode.
- **Help**
  - **SoFiA User Manual** – Open this user manual.
  - **What's This?** – Selecting this option and then clicking into any of the input fields and buttons will display a comprehensive help message.
  - **About SoFiA** – Display basic information about SoFiA.

← Previous ↑ Up Next →

User Interface

Tool Bar

For convenience, SoFiA comes with a tool bar that allows the user to access frequently needed actions with just a single mouse click. The tool bar can be hidden by unchecking the **Show Toolbar** option available either in the **Settings** menu or by right-clicking on the tool bar. The following actions are available from the tool bar:

- New File
- Open File
- Save File
- Run Pipeline
- Abort Pipeline
- Clear Messages
- What’s This?

← Previous ↑ Up Next →

User Interface

Pipeline Messages

The integrated pipeline message window just below SoFiA’s tool bar will show all *output messages* produced by the pipeline. These messages will typically convey status and progress information provided by the different components of the pipeline. The pipeline message window will also show *warnings* and *error messages*. These will be printed in red colour to make them stand out against the flow of regular progress messages. The pipeline message window is docked inside SoFiA’s main window and can be detached in order to save space on the desktop.

Once a pipeline run has finished, all output messages can be *saved to a log file* by selecting **Pipeline** → **Save Messages As...** from the menu bar. It is also possible to clear the pipeline display and *delete all previous messages* in preparation of a new pipeline run by selecting **Pipeline** → **Clear Messages** from the menu bar. Once deleted, messages cannot be recovered, so please ensure that you save all messages, if required, before clearing the display.

Progress Bar

The integrated progress bar beneath the pipeline display will show the progress made by the pipeline in percent. The progress bar does not reflect the actual time required to complete the pipeline run. Instead, the individual components of the pipeline, once completed, will count towards the total progress made. The progress bar will reach 100% upon successful completion of all components of the pipeline.

← Previous ↑ Up Next →

User Interface

Parameter Settings

Most of the lower part of the SoFiA window is occupied by the parameter input fields. These are subdivided into different *tabs*, each covering a particular category of settings in the pipeline process as outlined in the Parameters section of this document. The settings in each category can be accessed by clicking on the respective tab. In addition, *navigation buttons* have been provided at the bottom of the window to allow the user to conveniently cycle through all the tabs in their given order.

Each tab contains a number of grouped input *fields* and *buttons* that show the current parameter settings. Interacting with these fields and buttons with either the mouse or keyboard will allow the user to modify these settings. Settings will be remembered throughout a SoFiA session, but for permanent storage, parameter settings must be saved to disk using the **File** → **Save** or **File** → **Save As...** options from the menu.

← Previous ↑ Up Next →

User Interface

Status Bar

The status bar at the bottom of the SoFiA window is used to display short *status messages* from the software. This includes the success or failure of attempts to read or write parameter files, messages with regard to the pipeline status, etc. While most actions and events will display status information in the status bar, major issues will result in the launch of an additional *message window* to request user feedback or advise of serious problems.

← Previous ↑ Up Next →

Parameters

Introduction

The main purpose of SoFiA is to allow pipeline users to comfortably create and edit parameter settings for the pipeline. While parameter files can in principle be created and edited by hand, SoFiA is designed to make this task much easier by guiding the user through the set of available parameters and providing additional information and help on individual settings if required.

Format of Parameter Files

SoFiA parameter files are regular ASCII files with arbitrary file names that define a set of parameters controlling the source finding and parameterisation tasks carried out by the pipeline. Parameters are defined in the following human- and machine-readable form:

```
module.parameter = value
```

Each parameter setting must be on a separate line. An arbitrary number (including zero) of space and tabulator characters are allowed around the assignment operator (=) and at the beginning or end of each line; these will be ignored by the parser. Any empty lines and lines starting with a hash character (#) will be ignored as well and treated as a comment.

Available Parameters

The parameters of SoFiA are grouped into the following seven categories:

Category	Description
Input	Specification of the input data files.
Input Filter	A range of data filtering and preconditioning algorithms prior to running the source finding and parameterisation, including spatial and spectral smoothing and noise scaling.
Source Finding	Settings for the actual source finding algorithms.
Merging	Settings related to the merging of detections into sources.
Parameterisation	The settings for the source parameterisation algorithm used to measure the sources' observational and physical parameters.
Output Filter	A range of options for filtering the output catalogue based on the measured source parameters.
Output	The output data products and formats as well as the range of source parameters to be included in the output catalogue.

Please click on the individual categories for a detailed description of the available parameters and their possible values.

← Previous ↑ Up Next →

Parameters

Input

This section describes the different input data products and selections offered by SoFiA, including input data cubes, mask cubes, weights cubes and functions, subcube definition, data flagging, and source finding in regions around optical sources.

Input Files

SoFiA can read and process three types of input cubes, including the actual data cube to be searched, a mask cube of known sources and a weights cube to be applied prior to source finding.

Specifying an input data cube with the `import.inFile` parameter is mandatory, and SoFiA currently only supports FITS data cubes. Cubes stored in other data formats will have to be converted to FITS format before they can be processed by SoFiA. While SoFiA was originally designed to find sources in three-dimensional data cubes, it can also process two-dimensional images. In this case, the third axis is treated as being only a single pixel wide, and all settings regarding the third axis in the different processing options offered by SoFiA will have to be adjusted to be consistent with that assumption.

SoFiA can also read an initial mask cube, specified with the `import.maskFile` parameter, that marks the positions of known sources. This mask cube could, e.g., be from a previous run of SoFiA. This is useful in two situations: firstly, with source finding disabled, SoFiA can be used solely to parameterise sources at known positions as specified in the mask cube. Alternatively, SoFiA could be run multiple times with different source finding settings, in which case newly detected pixels would get added to any existing mask cube given in the input.

Using the `import.weightsFile` parameter, SoFiA can also read and apply a weights cube which must have the same dimensions as the input data cube by which it is multiplied prior to source finding. This is useful, e.g., to normalise noise levels across the data cube in cases where the noise varies across the cube in a known fashion. Alternatively, noise variations can also be corrected by applying an analytic function specified by the `import.weightsFunction` parameter. The three dimensions of the cube are referred to as `x`, `y` and `z`, and a number of common mathematical functions and operators are supported by SoFiA. For example, `0.7 * sin(z / 2.1)` would apply a sinusoidal correction to the third (spectral) axis. Note that the weights function will not be applied whenever a weights cube is specified.

Parameter:	**import.inFile**
Type:	`string`
Values:	
Default:	
Description:	Full path and name of input data cube. This option is mandatory, and there is no default. Note that only FITS files are currently supported by SoFiA.

Parameter:	**import.maskFile**
Type:	`string`
Values:	
Default:	
Description:	Name of an optional file containing a mask of pixels identified as part of a source, e.g. from a previous run of SoFiA. This can be used to re-parametrise sources without repeating the source finding step. The default is to not read a mask cube.

Parameter:	**import.weightsFile**
Type:	`string`
Values:	
Default:	
Description:	Name of an optional file containing weights of pixels in the input cube. The weights will be applied before running the source finder. The default is to not apply weights.

Parameter:	**import.weightsFunction**
Type:	`string`
Values:	
Default:	
Description:	Analytic function used to describe the data weights as a function of x, y, and z. The default is to

	not apply weights. The following mathematical functions from Numpy are supported: `sin()`, `cos()`, `tan()`, `arcsin()`, `arccos()`, `arctan()`, `arctan2()`, `sinh()`, `cosh()`, `tanh()`, `arcsinh()`, `arccosh()`, `arctanh()`, `exp()`, `log()`, `log10()`, `log2()`, `sqrt()`, `square()`, `power()`, `absolute()`, `fabs()`, and `sign()`. Note that the weights function is not applied whenever a weights cube is specified (see import.weightsFile).

Subcube

If the original input data cube is too large to fit into memory, or if only a small region of interest is to be processed, SoFiA can offer to only read and process a small subregion of the cube. After setting the `steps.doSubcube` parameter to `true`, the region can then be specified with the `import.subcube` parameter and can either be in pixels or in world coordinates as controlled by the `import.subcubeMode` parameter.

Parameter:	**steps.doSubcube**
Type:	`bool`
Values:	
Default:	`false`
Description:	If set to `true`, source finding can be carried out on a subcube to be defined by the import.subcube and import.subcubeMode options.

Parameter:	**import.subcube**
Type:	`list`
Values:	
Default:	`[]`
Description:	This parameter defines a subcube to be read in and processed by SoFiA. Depending on the value of import.subcubeMode, the range is either specified in pixels as `[x1, x2, y1, y2, z1, z2]` or in world coordinates as `[x, y, z, rx, ry, rz]` depending on the value of import.subcubeMode. In the latter case, `x`, `y` and `z` define the centre of the subcube, and `rx`, `ry` and `rz` specify the half-widths in the three dimensions. If world coordinates are used, all parameters must be in the native format as defined in the header of the data cube; e.g. if `CUNIT3` is `'Hz'` then both `z` and `rz` must be given in hertz. The default is an empty list, `[]`, which means to read the entire cube.

Parameter:	**import.subcubeMode**
Type:	`string`
Values:	`pixel`, `world`
Default:	`pixel`
Description:	This parameter defines whether import.subcube is specified in pixels (`pixel`) or in world coordinates (`world`).

Flagging

SoFiA can offer to flag some regions of the data cube that are either affected by radio-frequency interference or contain other artefacts or emission to be excluded, such as Galactic foreground emission. An arbitrary number of pixel/channel ranges can be flagged prior to source finding using the `flag.regions` parameter. The pixel values in flagged regions are marked as blank and thus excluded from all subsequent processing.

Parameter:	**steps.doFlag**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	Flag pixel and channel ranges before proceeding. Details are specified with the flag.regions option.

Parameter:	**flag.regions**
Type:	`list`
Values:	
Default:	`[]`
Description:	Pixel/channel range(s) to be flagged prior to source finding. Format: `[[x1, x2, y1, y2, z1, z2], ...]`. A place holder, `''` (two single quotes), can be used for the upper range limit (`x2`, `y2`, and `z2`) to flag all the way to the end, e.g. `[[0, '', 0, '', 0, 19]]` will flag the first 20 channels of the entire cube. The default is an empty list, `[]`, which means to not flag anything.

Optical Catalogue

Sometimes it is not desirable to carry out a blind search of an entire data cube, but only small regions around the positions of known sources need to be search. This could be useful, e.g., to search for HI signals in the vicinity of known optical sources. To facilitate such a search mode, SoFiA offers the possibility to read in a source catalogue and automatically search for sources in small subcubes around the positions defined in the input catalogue.

The spatial and spectral size of the subcubes can be set with the `optical.spatSize` and `optical.specSize` parameters, respectively. The input source catalogue is specified with the `optical.sourceCatalogue` parameter. Currently, SoFiA supports only CSV (comma-separated value) files as input catalogues, and they must contain at least four columns with the labels ‘id’, ‘ra’, ‘dec’ and ‘z’. All spatial and spectral coordinates must be given in the native units of the input cube, e.g. in degrees and hertz (the default units of equatorial coordinates and frequencies in FITS files).

SoFiA can either produce a separate output catalogue for each input position or alternatively merge all the individual output catalogues into a single catalogue. The behaviour can be controlled by the user with the `optical.storeMultiCat` parameter.

Parameter:	steps.doOptical
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, run SoFiA on multiple, smaller sub-cubes centred on positions defined in an input source catalogue. A catalogue file will need to be specified (see parameter optical.sourceCatalogue). This could, e.g., be an optical galaxy catalogue with the aim to search for HI detections at the positions of all galaxies.

Parameter:	optical.sourceCatalogue
Type:	`string`
Values:	
Default:	
Description:	This parameter defines the full path to the input catalogue required for catalogue-based source finding (see parameter steps.doOptical). There is no default.

Parameter:	optical.spatSize
Type:	`float`
Values:	≥ `0.0`
Default:	`0.01`
Description:	This parameter defines the spatial size of the sub-cube to be searched around each catalogue position. The size must be specified in the native units of the data cube, e.g. in degrees.

Parameter:	optical.specSize
Type:	`float`
Values:	≥ `0.0`
Default:	`1e+5`
Description:	This parameter defines the spectral size of the sub-cube to be searched around each catalogue position. The size must be specified in the native units of the data cube, e.g. in km/s or Hz.

Parameter:	optical.storeMultiCat
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, a separate output catalogue will be created for each input position, containing only the sources found in that subcube. In addition, a single, merged catalogue will also be created. By default this parameter is set to `false`, in which case only a single output catalogue file is generated.

← Previous ↑ Up Next →

Parameters

Input Filter

This section describes the different input filtering options offered by SoFiA. Currently, three different filters are available, including a simple, three-dimensional *smoothing* filter, a basic *noise-scaling* filter and a three-dimensional *wavelet decomposition* algorithm. Their settings are explained below.

Smoothing

This filter smoothes the data cube in all three dimensions by convolving with a specific kernel function selected by the user with the `smooth.kernel` parameter. Currently, Gaussian, boxcar and median filters are supported. The size of the kernel can be controlled independently in each of the three dimensions, using the `smooth.kernelX`, `smooth.kernelY` and `smooth.kernelZ` parameters.

Parameter:	**steps.doSmooth**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, spatially and spectrally smooth the data cube prior to source finding.

Parameter:	**smooth.kernel**
Type:	`string`
Values:	`gaussian`, `boxcar`, `median`
Default:	`gaussian`
Description:	Type of smoothing kernel used in both spatial and spectral smoothing. The possible options are `gaussian`, `boxcar` or `median`.

Parameter:	**smooth.edgeMode**
Type:	`string`
Values:	`constant`, `nearest`, `reflect`, `mirror`, `wrap`
Default:	`constant`
Description:	Pixel values assumed by the smoothing algorithm outside the boundaries of the cube. The following options are available: `constant`: assume a constant value of 0. `nearest`: assume a constant value equal to that of the nearest edge pixel. `reflect`: mirror values at the edge, thereby duplicating the edge pixel itself. `mirror`: mirror values at the centre of the outermost pixel, thereby avoiding duplication of the edge pixel itself. `wrap`: copy values from the opposite edge of the cube.

Parameter:	**smooth.kernelX**
Type:	`float`
Values:	≥ `0.0`
Default:	`3.0`
Description:	Kernel size in pixels for the first (spatial) coordinate. For Gaussian kernels the value refers to the FWHM.

Parameter:	**smooth.kernelY**
Type:	`float`
Values:	≥ `0.0`
Default:	`3.0`
Description:	Kernel size in pixels for the second (spatial) coordinate. For Gaussian kernels the value refers to the FWHM.

Parameter:	**smooth.kernelZ**
Type:	`float`
Values:	≥ `0.0`
Default:	`3.0`
Description:	Kernel size in pixels for the third (spectral) coordinate. For Gaussian kernels the value refers to

	the FWHM.

Noise Scaling

The purpose of this filter is to automatically normalise the data cube by the local noise level. This can be helpful in situations where the noise varies across the data cube, e.g. as a function of frequency, which would be problematic for source finding algorithms that assume a constant noise level. Noise scaling can be applied to each of the three cube dimensions independently. Three different algorithms for measuring the local noise level, specified with the `scaleNoise.statistic` parameter, are currently offered by SoFiA: standard deviation, median absolute deviation and fitting of a Gaussian function to the negative part of the flux histogram.

Parameter:	**steps.doScaleNoise**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, normalise noise levels across the data cube prior to source finding.

Parameter:	**scaleNoise.scaleX**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, apply noise normalisation in first (spatial) dimension.

Parameter:	**scaleNoise.scaleY**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, apply noise normalisation in second (spatial) dimension.

Parameter:	**scaleNoise.scaleZ**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, apply noise normalisation in third (spectral) dimension.

Parameter:	**scaleNoise.statistic**
Type:	`string`
Values:	`std`, `mad`, `negative`
Default:	`mad`
Description:	Statistic used to determine the noise in the data cube. The following options are available: standard deviation (`std`), median absolute deviation (`mad`) or fitting of a Gaussian function to the negative part of the flux histogram (`negative`).

Parameter:	**scaleNoise.edgeX**
Type:	`int`
Values:	≥ `0`
Default:	`0`
Description:	Size of the edge (in pixels) to be excluded in first (spatial) dimension.

Parameter:	**scaleNoise.edgeY**
Type:	`int`
Values:	≥ `0`
Default:	`0`
Description:	Size of the edge (in pixels) to be excluded in second (spatial) dimension.

Parameter:	**scaleNoise.edgeZ**
Type:	`int`
Values:	≥ `0`
Default:	`0`
Description:	Size of the edge (in pixels) to be excluded in third (spectral) dimension.

Wavelet Filter

The purpose of the wavelet filter is to carry out a wavelet decomposition of the entire data cube in all three dimensions, using the 2D–1D wavelet reconstruction algorithm of Flöer & Winkel (2012). By carefully selecting the range of spatial and spectral scales to be used in the decomposition, structure at certain, undesirable scales can be filtered out, e.g. the small-scale structure associated with statistical noise in the cube. The range of wavelet scales to be used in the reconstruction can be specified by the user with the `wavelet.scaleXY` and `wavelet.scaleZ` parameters. In addition, a relative flux threshold, specified with the `wavelet.threshold` parameter, is applied to each wavelet scale, and only those components above that flux threshold are included in the reconstructed cube.

Note that the wavelet filter does not do any source finding by itself, but simply generates a reconstructed wavelet cube. Any source finding must be done separately, e.g. by enabling SoFiA's threshold finder.

Parameter:	steps.doWavelet
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, decompose the data cube into wavelet components using the 2D–1D wavelet decomposition algorithm of Flöer & Winkel (2012).

Parameter:	wavelet.threshold
Type:	`float`
Values:	≥ `0.0`
Default:	`5.0`
Description:	Flux threshold used in the wavelet reconstruction processs in multiples of the rms noise. Note that this threshold only determines which wavelet components are added to the decomposed cube. Any source finder run after the reconstruction will use its own flux threshold.

Parameter:	wavelet.iterations
Type:	`int`
Values:	≥ `1`
Default:	`3`
Description:	Number of iterations to be used in the wavelet reconstruction process.

Parameter:	wavelet.scaleXY
Type:	`int`
Values:	≥ `-1`
Default:	`-1`
Description:	Number of *spatial* scales used in the decomposition. The default value of `-1` will automatically determine the appropriate number of scales based on the actual data cube.

Parameter:	wavelet.scaleZ
Type:	`int`
Values:	≥ `-1`
Default:	`-1`
Description:	Number of *spectral* scales used in the decomposition. The default value of `-1` will automatically determine the appropriate number of scales based on the actual data cube.

Parameter:	wavelet.positivity
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, include only positive wavelet components in the decomposition. Otherwise, negative components will be included as well.

← Previous ↑ Up Next →

Parameters

Source Finding

In this section the settings for the actual source finding algorithms are described. SoFiA offers three different algorithms: the *Smooth+Clip finder* (which is the default source finder in SoFiA), the *Characterised Noise HI finder*, and a simple *threshold finder*.

Smooth+Clip Finder

The Smooth+Clip (S+C) finder works by smoothing the data cube on different scales in all three dimensions, measuring the noise in each smoothed copy and then applying a relative flux threshold (in multiples of the noise level) to extract the significant pixels on each scale. A basic description of the algorithm can be found in Serra, Jurek & Flöer (2012). Several settings can be controlled by the user, including the range and type of filter kernels to be used in the smoothing (`SCfind.kernels`) as well as the flux threshold to be applied (`SCfind.threshold`).

Parameter:	**steps.doSCfind**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, run the smooth + clip finder.

Parameter:	**SCfind.threshold**
Type:	`float`
Values:	≥ `0.0`
Default:	`6.0`
Description:	Flux threshold relative to the noise level of the cube. The default value of `6.0` σ is likely to miss a large number of faint sources in most cases, and lower values will be required to maximise completeness.

Parameter:	**SCfind.edgeMode**
Type:	`string`
Values:	`constant`, `nearest`, `reflect`, `mirror`, `wrap`
Default:	`constant`
Description:	Pixel values assumed by the smoothing algorithm outside the boundaries of the cube. The following options are available: `constant`: assume a constant value of 0. `nearest`: assume a constant value equal to that of the nearest edge pixel. `reflect`: mirror values at the edge, thereby duplicating the edge pixel itself. `mirror`: mirror values at the centre of the outermost pixel, thereby avoiding duplication of the edge pixel itself. `wrap`: copy values from the opposite edge of the cube.

Parameter:	**SCfind.rmsMode**
Type:	`string`
Values:	`std`, `mad`, `negative`
Default:	`negative`
Description:	Statistic used to determine the noise in the data cube. The following options are available: standard deviation (`std`), median absolute deviation (`mad`) or fitting of a Gaussian function to the negative part of the flux histogram (`negative`).

Parameter:	**SCfind.kernelUnit**
Type:	`string`
Values:	`pixel`, `world`
Default:	`pixel`
Description:	This parameter defines whether the kernel parameters set by SCfind.kernels are specified in `pixel` or `world` coordinates.

Parameter:	**SCfind.kernels**
Type:	`list`

Values:	
Default:	`[[ 0, 0, 0,'b'],[ 0, 0, 3,'b'],[ 0, 0, 7,'b'],[ 0, 0, 15,'b'],[ 3, 3, 0,'b'],[ 3, 3, 3,'b'],[ 3, 3, 7,'b'],[ 3, 3, 15,'b'],[ 6, 6, 0,'b'],[ 6, 6, 3,'b'],[ 6, 6, 7,'b'],[ 6, 6, 15,'b']]`
Description:	List of kernels to be used for smoothing. The format is `[[dx, dy, dz, 'type'], ...]` where `dx`, `dy`, and `dz` are the spatial and spectral kernel sizes (FWHM), and `'type'` can be boxcar (`'b'`) or Gaussian (`'g'`). Note that `'type'` only applies to the spectral axis, and the spatial kernel is always Gaussian. Kernel sizes can be given either in pixels or in world coordinates depending of the value of SCfind.kernelUnit.

Parameter:	**SCfind.sizeFilter**
Type:	`float`
Values:	`0.0` – `1.0`
Default:	`0.0`
Description:	Size filtering; set to `0.0` to not size-filter (default and recommended setting). Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	**SCfind.maskScaleXY**
Type:	`float`
Values:	
Default:	`2.0`
Description:	Set already detected pixels to σ × `threshold` × `maskScaleXY` before spatial smoothing, where σ is the rms noise level. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	**SCfind.maskScaleZ**
Type:	`float`
Values:	
Default:	`2.0`
Description:	Set already detected pixels to σ × `threshold` × `maskScaleZ` before spectral smoothing, where σ is the rms noise level. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	**SCfind.verbose**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, print additional progress information. Note that this is a *hidden* option not accessible through the graphical user interface.

Characterised Noise HI Finder

The Characterised Noise HI (CNHI) finder was developed by Jurek (2012) and uses the Kuiper test to identify regions of the data cube whose flux distribution is inconsistent with that of pure, statistical noise. There are currently no additional settings to control the CNHI finder.

Parameter:	**steps.doCNHI**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, run the Characterised Noise HI (CNHI) source finder by Jurek (2012).

Parameter:	**CNHI.pReq**
Type:	`float`
Values:	≥ `0`
Default:	`1e-5`
Description:	Minimum probability requirement for detections to be considered genuine. Sensible values are typically in the range of about 10^{-3} to 10^{-5}.

Parameter:	**CNHI.qReq**
Type:	`float`
Values:	≥ `1.0`

Default:	3.8
Description:	This is the Q value of the Kuiper test, which is a heuristic parameter for assessing the quality/accuracy of the probability calculated from the Kuiper test. The minimum scale that the CNHI source finder processes is increased until it is sufficiently large to ensure that the required Q value is achieved. This requirement supersedes the user-specified minimum scale (see parameter CNHI.minScale). The default value is 3.8, which is the generally accepted minimally useful value.

Parameter:	**CNHI.minScale**
Type:	int
Values:	≥ 1
Default:	5
Description:	The minimum size of test regions.

Parameter:	**CNHI.maxScale**
Type:	int
Values:	≥ -1
Default:	-1
Description:	The maximum size of test regions. The default value of -1 is a flag value that sets the maximum size to half of the size of the spectral axis.

Parameter:	**CNHI.medianTest**
Type:	bool
Values:	true, false
Default:	true
Description:	This parameter determines whether test regions need to have a median greater than that of the remaining data in order to be considered a source.

Parameter:	**CNHI.verbose**
Type:	int
Values:	0 – 2
Default:	1
Description:	An integer value that indicates the level of verbosity of the CNHI finder. Values of 0, 1 and 2 correspond to **none**, **minimal** and **maximal**, respectively.

Threshold Finder

The threshold finder is the most basic type of source finding algorithm in SoFiA. Using the threshold.threshold parameter, it simply applies a flux threshold (either relative or absolute as defined by threshold.clipMethod) to the data and considers all pixels above that threshold to be significant. The threshold finder is not particularly powerful on its own, but is usually applied in combination with an additional input filter such as the 2D–1D wavelet decomposition algorithm.

Parameter:	**steps.doThreshold**
Type:	bool
Values:	true, false
Default:	false
Description:	If set to true, run the threshold finder.

Parameter:	**threshold.threshold**
Type:	float
Values:	≥ 0.0
Default:	4.0
Description:	Absolute or relative flux threshold (see threshold.clipMethod). The default value of 4.0 σ is likely to miss a large number of faint sources in most cases, and lower values will be required to maximise completeness.

Parameter:	**threshold.clipMethod**
Type:	string
Values:	relative, absolute
Default:	relative

Description:	Is the threshold specified by threshold.threshold `relative` to the noise level or in `absolute` flux units?

Parameter:	**threshold.rmsMode**
Type:	`string`
Values:	`std`, `mad`, `negative`
Default:	`std`
Description:	Statistic used to determine the noise in the data cube. The following options are available: standard deviation (`std`), median absolute deviation (`mad`) or fitting of a Gaussian function to the negative part of the flux histogram (`negative`).

Parameter:	**threshold.verbose**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, print additional progress information. Note that this is a *hidden* option not accessible through the graphical user interface.

← Previous ↑ Up Next →

Parameters

Merging

The settings below control how significant pixels detected by any of SoFiA's source finding algorithms get merged into final sources.

Merging

Any source finding algorithm implemented in SoFiA will not create a final source catalogue, but instead produce a binary mask in which all pixels that are considered as significant are marked. In order to obtain a final source list, those pixels will need to be merged into sources, which is the purpose of the merging module.

Merging of pixels into sources is controlled by two sets of parameters. The first set defines the maximum separation that two detected pixels are allowed to have in order to be considered as part of the same source. This maximum separation is controlled independently for each dimension, using the `merge.radiusX`, `merge.radiusY` and `merge.radiusZ` parameters. Once all pixels have been merged into sources, the size of each source is checked (again independently in each dimension), and sources falling below the size thresholds (specified with `merge.minSizeX`, `merge.minSizeX` and `merge.minSizeX`) are discarded. This allows many spurious detections, such as bright noise peaks, to be eliminated based on their small size.

Parameter:	**steps.doMerge**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, pixels detected by the source finder will be merged into final sources based on user-defined separation and size criteria.

Parameter:	**merge.radiusX**
Type:	`int`
Values:	≥ `0`
Default:	`3`
Description:	Merging radius in first dimension in pixels. Note that a value of `0` means that no merging takes place and each detected pixel is retained as a separate source.

Parameter:	**merge.radiusY**
Type:	`int`
Values:	≥ `0`
Default:	`3`
Description:	Merging radius in second dimension in pixels. Note that a value of `0` means that no merging takes place and each detected pixel is retained as a separate source.

Parameter:	**merge.radiusZ**
Type:	`int`
Values:	≥ `0`
Default:	`3`
Description:	Merging radius in third dimension in pixels. Note that a value of `0` means that no merging takes place and each detected pixel is retained as a separate source.

Parameter:	**merge.minSizeX**
Type:	`int`
Values:	≥ `1`
Default:	`3`
Description:	Minimum required extent in first dimension of genuine sources after merging. Sources below this size will be discarded.

Parameter:	**merge.minSizeY**
Type:	`int`
Values:	≥ `1`
Default:	`3`

Description:	Minimum required extent in second dimension of genuine sources after merging. Sources below this size will be discarded.

Parameter:	**merge.minSizeZ**
Type:	`int`
Values:	≥ `1`
Default:	`2`
Description:	Minimum required extent in third dimension of genuine sources after merging. Sources below this size will be discarded.

← Previous ↑ Up Next →

Parameters

Parameterisation

The settings in this section control the way in which the observational parameters of detected sources are calculated. In addition, SoFiA offers the possibility of estimating the reliability of each individual detection under certain circumstances.

Parameterisation

Parameterisation is an important step of any source finding exercise, and the quality of the source parameters measured by SoFiA will determine the accuracy of any scientific analysis based on those parameters. In principal, SoFiA measures all parameters by integrating over the entire sources mask derived in the merging step, and the accuracy of the parameterisation will therefore depend of how accurate the source mask was determined by the combinations of source finding and merging.

Simple threshold finders have the tendency to produce masks that are too small and miss some of the faint outer parts of sources that are below the detection threshold. In order to rectify this problem, SoFiA offers two methods of optimising the source mask. Both methods work by iteratively growing an initially small mask outwards until the integrated flux within the mask does not increase any further. The first methods, enabled with the `parameters.optimiseMask` parameter, uses an ellipse in the spatial domain and a constant size in the spectral domain as the mask shape. It is therefore particularly useful for sources that have approximately the shape of an elliptical cylinder in a three-dimensional data cube. The second method, enabled with the `parameters.dilateMask` parameter, instead dilates the original mask in all three dimensions and is therefore more appropriate for sources of arbitrary shape, such as galaxies that are both spatially and spectrally well-resolved.

Another option offered by SoFiA is the possibility to fit a Busy Function (Westmeier et al. 2014) to the integrated spectrum of each source and determine basic source parameters from that fit. This is particularly useful for galaxies that are spatially unresolved. Parameterisation based on Busy Function fitting is usually more accurate than direct measurement of parameters such as peak flux or line width, because the fit is less sensitive to individual noise peaks in the data that would otherwise influence the measurement. Busy function fitting can be enabled with the `parameters.fitBusyFunction` parameter, and source parameters derived from the fit will be included in the output catalogue in addition to the ‘traditional’ measurements of the same parameters.

Parameter:	**steps.doParameterise**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, run the mask optimisation and source parameterisation module.

Parameter:	**parameters.optimiseMask**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	Run the mask optimisation algorithm based on fitting and growing ellipses to achieve more accurate flux measurements. Note that the improved integrated fluxes obtained by this algorithm will come at the cost of increased statistical uncertainties in most source parameters (also see parameters.dilateMask).

Parameter:	**parameters.dilateMask**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	Run the mask optimisation algorithm based on spatially dilating the initial mask to achieve more accurate flux measurements. The advantage of this algorithm is that it preserves the source shape (also see parameters.optimiseMask).

Parameter:	**parameters.dilateThreshold**
Type:	`float`
Values:	
Default:	`0.02`

Description:	No idea what this parameter does... Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	parameters.dilatePixMax
Type:	`int`
Values:	
Default:	`10`
Description:	No idea what this parameter does... Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	parameters.dilateChan
Type:	`int`
Values:	
Default:	`1`
Description:	No idea what this parameter does... Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	parameters.fitBusyFunction
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	Fit the Busy Function (Westmeier et al. 2014) to the integrated spectrum of each source for more accurate parameterisation.

Reliability

SoFiA can automatically estimate the reliability of individual sources and discard unreliable sources in certain circumstances, using the method introduced by Serra, Jurek & Flöer (2012). The method works by not only detecting and parameterising sources with positive flux, but also 'pseudo-sources' with negative flux. Under the assumption that all negative sources are artefacts (e.g. noise peaks), one can then estimate the reliability of positive sources by comparing the regions of parameter space occupied by positive and negative sources. Simply speaking, a positive source located in a region of parameter space that is occupied by numerous negative sources is less likely to be genuine than a positive source located in a region of parameter space that is free of negative detections.

In order for the reliability calculation to work and be accurate, a few conditions have to me met. Firstly, all noise and artefacts in the data cube must be centred on zero such that both positive and negative artefacts exist at a ratio of approximately 1:1. Secondly, all genuine sources in the data cube must have positive flux (e.g., no absorption signals). Finally, the threshold of the source finder must be set to a fairly low value to ensure that a substantial number of negative (and positive) noise peaks and artefacts get detected. This is to ensure that the density of negative detections in parameter space is sufficiently high to be accurately measured.

Several aspects of reliability calculation can be controlled by the user, including the smoothing kernel to be used in logarithmic parameter space (`reliability.kernel`) and the reliability threshold (`reliability.threshold`) to be used to discard all sources whose reliability is below that threshold. The optimal kernel size can also be determined automatically within SoFiA by setting the `reliability.autoKernel` parameter to `true` (this is the default behaviour).

Parameter:	steps.doReliability
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, use negative detections to determine the reliability of each source based on the algorithm of Serra, Jurek & Flöer (2012).

Parameter:	reliability.threshold
Type:	`float`
Values:	`0.0` – `1.0`
Default:	`0.9`
Description:	Discard all sources whose reliability is below this threshold.

Parameter:	reliability.kernel

Type:	`list`
Values:	
Default:	`[0.15, 0.05, 0.1]`
Description:	Size of the 3D smoothing kernel in log(parameter) space (see reliability.parSpace). This parameter will be ignored if reliability.autoKernel is set to `true` (default behaviour).

Parameter:	**reliability.autoKernel**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	This parameter controls whether the kernel size to be used for reliability calculation should be determined automatically (`true`) or manually (`false`). Default is `true`. If set to `false`, the reliability.kernel parameter must be used to specify the kernel size. Also see the reliability.negPerBin and reliability.skellamTol parameters.

Parameter:	**reliability.makePlot**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, a PDF file showing the distribution of positive and negative detections in parameter space will be created for diagnostic purposes.

Parameter:	**reliability.parSpace**
Type:	`list`
Values:	
Default:	`['ftot', 'fmax', 'nrvox']`
Description:	Defines the 3D parameter space in which to determine reliability. It is strongly recommended to use the pre-defined default settings. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	**reliability.fMin**
Type:	`float`
Values:	≥ `0.0`
Default:	`0.0`
Description:	Minimum total flux for a source to be considered reliable. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	**reliability.negPerBin**
Type:	`float`
Values:	≥ `1.0`
Default:	`5.0`
Description:	This parameter defines the minimum number of negative detections per bin in parameter space required for the automatic kernel size determination. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	**reliability.skellamTol**
Type:	`float`
Values:	
Default:	`-0.2`
Description:	Tolerance parameter for reliability kernel size determination. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameters

Output Filter

In the future, this module will offer the possibility for the user to define parameter ranges for reliable detections. All sources outside those ranges will be removed from the output catalogue. This module has not yet been included in SoFiA, and output filtering is not currently possible.

← Previous ↑ Up Next →

Parameters

Output

The parameters described below control the creation and storage of source catalogues and other data products, including cubes and moment maps as well as more sophisticated data products such as integrated spectra and position–velocity diagrams.

Output Files

The primary output product of SoFiA are source catalogues. Several formats are offered, including human-readable **ASCII** files and VO tables in **XML** format. Creation of **SQL** files for insertion of source parameters into SQL databases is planned for the future, but has not yet been implemented.

In addition to catalogues, SoFiA can produce imaging products, including a copy of the filtered input cube (assuming that an input filter was applied), a copy of the source mask cube, moment-0 and moment-1 maps of all detected pixels, as well as individual imaging products for each detected source (including a small subcube, moment-0, 1 and 2 images, a position–velocity diagram, and an integrated spectrum). All of these output products can be compressed using [gzip](#) if desired.

The names of all output files are normally generated by extracting the name of the input data cube and appending a specific identifier. For example, if the input data cube is named `datacube.fits`, ASCII and XML catalogues will be called `datacube_cat.ascii` and `datacube_cat.xml`, etc. If for some reason this is not desired, a different base name can be specified with the `writeCat.baseName` parameter. Note that any existing output files from a previous run of SoFiA will be **overwritten without warning**, so please ensure that any files which are still required are copied or renamed before running SoFiA again.

By default, all output files will be written to the directory where the input data cube is located. If this is impossible or undesired (e.g. because that directory is write-protected), a different output directory can be specified with the `writeCat.outputDir` parameter.

Parameter:	steps.doWriteCat
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, write output catalogue(s) to disk. Note that this is a *hidden* option not accessible through the graphical user interface.

Parameter:	writeCat.baseName
Type:	`string`
Values:	
Default:	
Description:	Optional base name of all output files. If not specified, the input file name will be used by default.

Parameter:	writeCat.outputDir
Type:	`string`
Values:	
Default:	
Description:	Optional directory path to which all output files are written. If not specified, the directory of the input data cube will be used by default.

Parameter:	writeCat.writeASCII
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, write catalogue in human-readable ASCII format.

Parameter:	writeCat.writeXML
Type:	`bool`
Values:	`true`, `false`
Default:	`false`

Description:	If set to `true`, write catalogue in VO table format (XML).

Parameter:	**writeCat.writeSQL**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, write catalogue in SQL format for insertion into an SQL database (not yet implemented).

Parameter:	**steps.doWriteFilteredCube**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, save a copy of the filtered data cube. Note that this will only work if at least one of the input filters was applied.

Parameter:	**steps.doWriteMask**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, save the mask cube.

Parameter:	**steps.doMom0**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, a moment-0 map of all detected sources will be created and saved to disk.

Parameter:	**steps.doMom1**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, a moment-1 map of all detected sources will be created and saved to disk.

Parameter:	**steps.doCubelets**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, a range of data products for each individual source will be created and written to disk, including a small *sub-cube*, *moment-0*, *1* and *2* maps, a *position–velocity diagram* along the morphological major axis (as derived from an ellipse fit to the moment-0 image) and an *integrated spectrum* of the source.

Parameter:	**writeCat.compress**
Type:	`bool`
Values:	`true`, `false`
Default:	`false`
Description:	If set to `true`, use gzip to compress all output files.

Parameter:	**writeCat.overwrite**
Type:	`bool`
Values:	`true`, `false`
Default:	`true`
Description:	If set to `true`, any existing output files will automatically get overwritten. If set to `false`, the pipeline will print an error message for any output file that already exists and skip writing that file.

Output Parameters

By default, SoFiA will write all measured source parameters into the output catalogue. This behaviour can be changed, however, by explicitly listing the desired output parameters with the `writeCat.parameters` option. Note

that any parameters that are either unknown or not measured will automatically be ignored by SoFiA.

Parameter:	**writeCat.parameters**
Type:	`list`
Values:	
Default:	`['*']`
Description:	List of parameters to appear in source catalogue. Format: `['par1', 'par2', ...]`. An asterisk, `['*']`, means that all parameters are written to the output catalogue.

← Previous ↑ Up Next →

Running the Pipeline

From the User Interface

Once an input data cube has been selected and the desired parameter settings have been chosen, the pipeline can be launched from within the user interface, either by selecting **Pipeline** → **Run Pipeline** from the menu or by clicking on the corresponding tool bar button. The pipeline run can be aborted at any time by selecting **Pipeline** → **Abort Pipeline** from the menu or by clicking on the respective tool bar button.

SoFiA will automatically save the current parameter settings to a temporary session file in the current working directory (named `SoFiA.session`) before starting the pipeline. Note that this session file will automatically be loaded again when exiting and restarting the SoFiA user interface from within the same working directory, so users can continue to work from where they previously stopped. Nonetheless, it is strongly advisable to explicitly save the final set of parameter settings into a permanent file for later use, as the session file is only temporary and will regularly be updated.

From the Terminal

Alternatively, the pipeline can be launched from the terminal after the parameter settings have been saved to an output file. To achieve this, open a terminal window and navigate to the directory where the output parameter file was saved. Then run the pipeline by typing

```
sofia_pipeline.py <parameter_file>
```

where `<parameter_file>` is the name of the SoFiA parameter file. There is no difference between running the pipeline from the user interface or from the terminal, but the latter is more useful in situations where the pipeline needs to be launched by another application or shell script, e.g. to automatically process a larger number of data cubes.

← Previous ↑ Up Next →

Output Products

SoFiA provides a large range of different output products as detailed below. All products will be saved either to the directory containing the input data cube or a specific output directory provided by the user. The user can also choose whether existing output files should automatically be overwritten or not.

Catalogues

SoFiA currently offers two different catalogue formats:

- A simple **ASCII file** that contains the source catalogue in human-readable format. Each source is listed in a separate line, while the columns list the individual source parameters. The file also contains a few header lines with additional information such as parameters names and units.
- An **XML file** that contains the source catalogue in VOTable format to be used in Virtual Observatory tools such as TOPCAT.

Additional Data Products

In addition to source catalogues, SoFiA can also generate a range of other useful data products. These can be selected in the output tab and include:

- A copy of the **filtered data cube** after application of all requested input filters. Note that this will only be different from the input data cube if at least one filter is applied to the data.
- A **mask cube** in FITS format that contains a pixel mask of all sources identified by SoFiA. The mask contains integer numbers that are equal to the unique source identification number. Mask files can also be read into SoFiA again to facilitate the parameterisation of sources that were detected in a previous source finding run.
- Maps of the **0^{th}** and **1^{st} moment** of all detections in FITS format. These are integrated across all pixels masked as part of a source.
- A map showing the total number of detected **channels** for each spatial pixel. (Note that this cannot currently be selected, but is automatically created whenever a moment-0 or 1 map is requested.)
- Small subcubes, so-called **cubelets**, of each individual source detected by SoFiA. This also includes **0^{th}**, **1^{st}** and **2^{nd} moment** maps created from the cubelet, a **position–velocity diagram** along the morphological major axis of each source, and an **integrated spectrum** of the source. All these individual products will be stored in a separate sub-folder named `objects`. Maps and cubes are stored in FITS format, while the integrated spectrum is stored as a plain text file.

← Previous ↑ Up Next →

References

Several of the source finding and parameterisation algorithms employed by SoFiA have been published in a special issue of the Publications of the Astronomical Society of Australia (PASA) on Source Finding and Visualisation. Selected PASA papers relevant to SoFiA have been included in the list of references below.

SoFiA Overview Paper

- SoFiA: a flexible source finder for 3D spectral line data
  Serra, P., Westmeier, T., Giese, N., Jurek, R., Flöer, L., Popping, A., Winkel, B., van der Hulst, T., Meyer, M., Koribalski, B. S., Staveley-Smith, L., Courtois, H., 2015, MNRAS, 448, 1922

Source Finding Algorithms

- 2D–1D Wavelet Reconstruction as a Tool for Source Finding in Spectroscopic Imaging Surveys
  Flöer, L., Winkel, B., 2012, PASA, 29, 244
- The Characterised Noise H I Source Finder: Detecting H I Galaxies Using a Novel Implementation of Matched Filtering
  Jurek, R., 2012, PASA, 29, 251
- Examining Alternatives to Wavelet Denoising for Astronomical Source Finding
  Jurek, R., Brown, S., 2012, PASA, 29, 352
- Using Negative Detections to Estimate Source-Finder Reliability
  Serra, P., Jurek, R., Flöer, L., 2012, PASA, 29, 296
- Duchamp: A 3D Source Finder for Spectral-line Data
  Whiting, M., 2012, MNRAS, 421, 3242
- Source-Finding for the Australian Square Kilometre Array Pathfinder
  Whiting, M., Humphreys, B., 2012, PASA, 29, 371

Source Parameterisation Algorithms

- The busy function: a new analytic function for describing the integrated 21-cm spectral profile of galaxies
  Westmeier, T., Jurek, R., Obreschkow, D., Koribalski, B. S., Staveley-Smith, L., 2014, MNRAS, 438, 1176
- The Extragalactic Distance Database: All Digital H I Profile Catalog
  Courtois, H. M., Tully, R. B., Fisher, J. R., Bonhomme, N., Zavodny, M., Barnes, A., 2009, AJ, 138, 1938

Source Finder Testing

- Comparison of Potential ASKAP H I Survey Source Finders
  Popping, A., Jurek, R., Westmeier, T., Serra, P., Flöer, L., Meyer, M., Koribalski, B. S., 2012, PASA, 29, 318
- Basic Testing of the Duchamp Source Finder
  Westmeier, T., Popping, A., Serra, P., 2012, PASA, 29, 276

Credits and Disclaimer

Authors

SoFiA Pipeline: Lars Flöer, Nadine Giese, Russell Jurek, Martin Meyer, Attila Popping, Paolo Serra, Tobias Westmeier, Benjamin Winkel
SoFiA User Interface: Tobias Westmeier

Contact

Tobias Westmeier
ICRAR M468
The University of Western Australia
35 Stirling Highway
Crawley WA 6009
Australia

E-mail: `tobias.westmeier [at] uwa.edu.au`

Licence and Disclaimer

Copyright